LAYER POTENTIAL EVALUATIONS ON DISTRIBUTED MEMORY MACHINES

BY

HAO GAO

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2020

Urbana, Illinois

Adviser:

Professor Andreas Klöckner

ABSTRACT

One of the main challenges of using integral equation methods (IEM) for solving partial differential equations is evaluating layer potentials with singular kernels. Quadrature by Expansion (QBX) is a quadrature method to evaluate such layer potentials accurately for targets near or on the source boundary, by forming expansions in the high-accuracy region away from the boundary, and evaluating the targets using the expansions. Recently, a new algorithm, called 'GIGAQBX', has combined QBX with the Fast Multipole Method to achieve linear complexity in terms of the number of degrees of freedom. Despite this advancement, QBX is still computationally expensive. To enable IEM on large-scale problems, this thesis investigates evaluating layer potentials on distributed-memory machines. The distributed algorithm introduced in this thesis is based on GIGAQBX and shows GIGAQBX contains plenty of parallelism. We evaluate our algorithm on the Comet supercomputer at the San Diego Supercomputer Center and show that it exhibits good strong scaling up to 1536 cores.

TABLE OF CONTENTS

CHAPTER 1: INTEGRAL EQUATION METHODS

Suppose we would like to solve an interior boundary value problem with a homogeneous PDE

$$Lu(x) = 0 \qquad \text{for } x \in \Omega \setminus \partial\Omega \tag{1.1}$$

$$\lim_{y \to x_-} u(y) = g(x) \qquad \text{for } x \text{ on the boundary of } \Omega\ , \tag{1.2}$$

where L is a linear, constant-coefficient, elliptic differential operator, and $\lim_{y \to x_{+/-}}$ represents the limit at x approaching from exterior/interior. Integral Equation Methods (IEMs) represent the solution u as layer potentials. For example,

$$u(x) = \int_{\partial\Omega} G(x,y)\sigma(y)dy, \tag{1.3}$$

where G is the free space Green's function associated with L. Then the PDE (1.1) is automatically satisfied at any point x away from $\partial\Omega$, following the definition of Green's function and Dirac delta function δ

$$Lu(x) = \int_{\partial\Omega} LG(x,y)\sigma(y)dy = \int_{\partial\Omega} \delta(x-y)\sigma(y)dy = 0. \tag{1.4}$$

The representation used in (1.3) is one example of layer potentials, called a Single Layer Potential. Commonly used layer potentials include

$$\text{the Single Layer Potential} \qquad (S\sigma)(x) := \int_\Gamma G(x,y)\sigma(y)dy, \tag{1.5}$$

$$\text{and the Double Layer Potential} \qquad (D\sigma)(x) := \int_\Gamma \frac{\partial}{\partial \hat{n}_y} G(x,y)\sigma(y)dy. \tag{1.6}$$

Here Γ is a smooth closed contour. The one-sided limits of these layer potentials [1] are

$$\lim_{y \to x_\pm} (S\sigma)(y) = (S\sigma)(x), \tag{1.7}$$

$$\lim_{y \to x_\pm} (D\sigma)(y) = \mp\frac{1}{2}\sigma(x) + (D\sigma)(x). \tag{1.8}$$

To solve the boundary value problem, IEMs only need to enforce the boundary condition

(1.2) by solving the integral equation

$$\lim_{y \to x_-} u(y) = \int_{\partial\Omega} G(x,y)\sigma(y)dy = g(x) \tag{1.9}$$

on the boundary $x \in \partial\Omega$. By contrast, Finite Difference Methods (FDMs) and Finite Element Methods (FEMs) need to discretize and solve for degrees of freedom at both the boundary and the interior of the domain. This means the number of degrees of freedom has adverse scaling with respect to the domain size and resolution compared to IEMs. This is a major advantage for large-scale applications as the number of discretization points can quickly become very large.

To solve the integral equation generated by the IEM (1.9), we use the Generalized Minimal Residual Method (GMRES) [2]. GMRES is a widely-used iterative algorithm for solving linear systems by repeatedly evaluating matrix-vector product in each iteration. When solving (1.9), the matrix-vector product corresponds to evaluating the layer potentials on the source surface. After the density function σ is found, we need to evaluate layer potentials at target points in Ω to get the solution to the boundary value problem. Therefore, it is critical to evaluate layer potentials accurately and efficiently for both solving and evaluating.

There are two main challenges for evaluating layer potentials like (1.5) and (1.6). First, the kernel G is usually singular or near-singular close to the boundary Ω, posing a challenge for accuracy. Second, the potential of a target point receives contribution from all sources, so the linear system generated by IEM is dense, posing a challenge for efficiency. To accurately evaluate layer potentials with singular kernels, we use Quadrature by Expansion (QBX). QBX recovers the accuracy by forming an expansion (e.g. Taylor expansion) away from the boundary in the high-accuracy region, and then computing the on-surface value using this expansion. QBX will be covered in more detail in Section 2.1. In terms of efficiency, QBX can be combined with fast algorithms, for example variants of Fast Multipole Method (FMM), to reduce the cost of dense matrix-vector product in each iteration from $O(MN)$ to $O(M+N)$ where M is the number of targets and N is the number of sources, arising from discretization of the source surface into quadrature nodes. The combination of FMM and QBX will be covered in Section 2.2. However, even if FMM acceleration is used, large-scale layer potential evaluations can still be prohibitively expensive on a single machine. Chapter 3 will show an efficient algorithm for evaluating layer potentials on distributed-memory systems.

Related work. For distributed FMM, [3] proposes partitioning the boxes among ranks along a space-filling curve, forming a local tree on each rank, and computing the potential independently. Note that this thesis uses the term "ranks" to mean MPI ranks. [4] extends [3]'s algorithm by proposing a new multipole communication algorithm and implementing

the local operators on GPUs. The contribution of this thesis is to extend the distributed FMM algorithm into evaluating layer potentials. The distributed algorithm proposed is highly influenced by [3] and [4].

CHAPTER 2: QUADRATURE BY EXPANSION

2.1 MATHEMATICAL FORMULATION OF QBX

Suppose we would like to evaluate the single layer potential

$$u(x) = \int_\Gamma G(x,y)\sigma(y)dy, \tag{2.1}$$

on a closed contour Γ with G being the Green's function of Helmholtz equation, i.e.

$$G(x,y) = \frac{i}{4}H_0^{(1)}(k|x-y|), \tag{2.2}$$

where $H_l^{(1)}$ denotes the Hankel function of the first kind of order l. Figure 2.1 shows the error behavior when evaluating (2.1) with Gaussian quadrature on a circular contour Γ. Note that $H_l^{(1)}$ has a singularity at 0. When x is far away from the boundary Γ, $H_l^{(1)}$ is a smooth function, and Gaussian quadrature can compute (2.1) accurately. However, when x is close or on the boundary, both plots in Figure 2.1 show a "band" region of inaccuracy around Γ. Comparing Figure 2.1a to Figure 2.1b suggests increasing quadrature counts makes the "band" region of inaccuracy tighter, but cannot mitigate the singularity when x is on the boundary Γ.

Following Graf's addition theorem [5, Eq. 10.23.7], we have

$$H_0^{(1)}(k|x-y|) = \sum_{l=-\infty}^{\infty} H_l^{(1)}(k|y-c|)e^{il\theta'}J_l(k|x-c|)e^{-il\theta}, \tag{2.3}$$

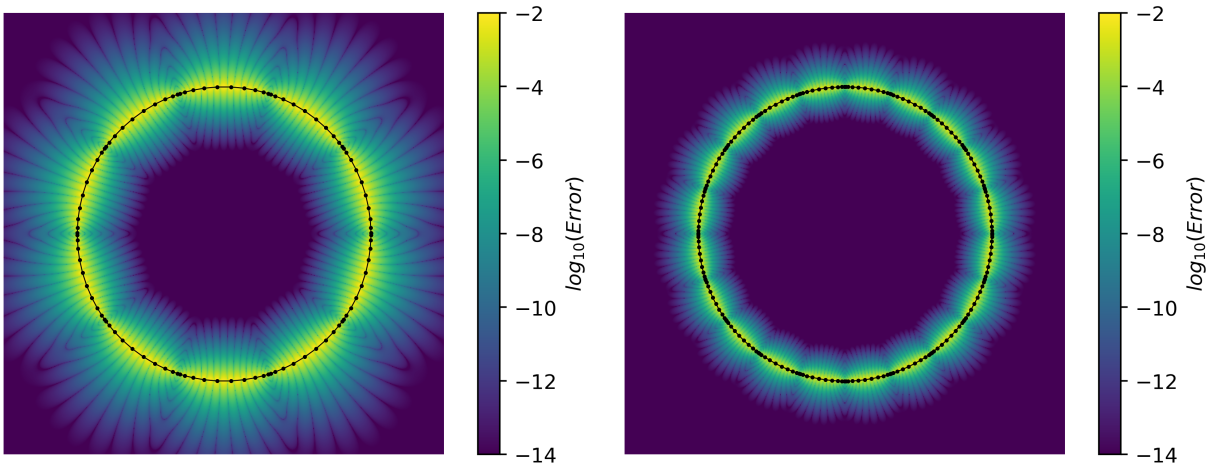
where J_l is the Bessel function of order l, θ is polar coordinate of x with respect to c, and θ' is polar coordinate of y with respect to c, as shown in Figure 2.2. Substitute (2.3) into (2.1), we have

$$u(x) = \sum_{l=-\infty}^{\infty} \alpha_l J_l(k|x-c|)e^{-il\theta}, \tag{2.4}$$

where

$$\alpha_l = \int_\Gamma \frac{i}{4}H_l^{(1)}(k|y-c|)e^{il\theta'}\sigma(y)dy. \tag{2.5}$$

Note that for c far away from the boundary Γ, $H_l^{(1)}(k|y-c|)$ is a smooth function and the integral can be evaluated accurately using the Gaussian quadrature. *Quadrature by Expansion* (QBX) [6] takes advantages of this observation. When evaluating layer potentials



(a) 10 panels and 10 quadrature points per panel (b) 20 panels and 10 quadrature points per panel

Figure 2.1: These two figures show the error of evaluating the Single Layer Potential (2.1) with Gaussian quadrature. For both figures, the contour Γ is a circle, and we choose $k = 0.5$. For the left figure, the contour is divided evenly to 10 panels and Gaussian quadrature with 10 quadrature points is performed on each panel. The total number of quadrature nodes is therefore 100. The right figure follows the same strategy, but uses 20 panels instead of 10 panels, so the total number of quadrature nodes is 200.

with singular kernels at target points near or on the boundary, QBX chooses an expansion center c in the high-accuracy region of Figure 2.1 away from the boundary Γ, and evaluates α_l in (2.5) using Gaussian quadrature. The potential at a point x near or on the boundary can then be approximated by truncating (2.4),

$$u(x) \approx \sum_{l=-p}^{p} \alpha_l J_l(k|x-c|)e^{-il\theta}. \tag{2.6}$$

The truncated order p in (2.6) is called the *QBX order*. Although we are using Helmholtz kernel and Gaussian quadrature as an example, QBX can be used for evaluating other singular kernels, with other high-order quadrature methods, following the same strategy.

Compared to using Gaussian quadrature directly when evaluating the potentials of targets near the boundary, Figure 2.3 shows QBX significantly improves the accuracy for targets inside or on the boundary of the circle centered at c with radius $\min_{x'\in\Gamma} |x'-c|$ [6]. Moreover, the following theorem shows the error of QBX for targets on the boundary:

Theorem 2.1 ([6, Theorem 1])**.** *Suppose that Γ is a smooth, bounded curve embedded in $\mathbb{R}^2$, that $B_r(c)$ is the ball of radius r about c, and that $B_r(c) \cap \Gamma = \{x\}$. Let Γ be divided into M panels, each of length h and let q be a non-negative integer that defines the number*

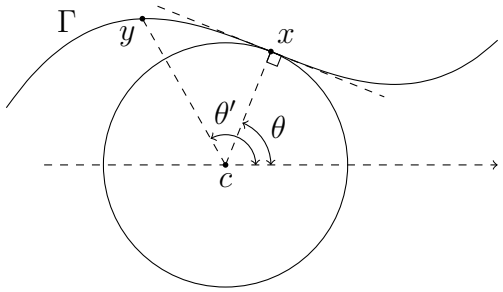


Figure 2.2: QBX expansion center and Graf's addition theorem.

of nodes of the smooth Gaussian quadrature used to compute the coefficients α_l^{QBX} *according to the formula* (2.5). *For* $0 < \beta < 1$, *there are constants* $C_{p,\beta}$ *and* $\widetilde{C}_{p,q,\beta}$ *so that if* σ *lies in the Hölder space* $\mathcal{C}^{p,\beta}(\Gamma) \cap \mathcal{C}^{2q,\beta}(\Gamma)$, *then*

$$\left|(S\sigma)(x) - \sum_{l=-p}^{p} \alpha_l^{QBX} J_l(k|x-c|)e^{-il\theta}\right| \leq \underbrace{C_{p,\beta} r^{p+1} \|\sigma\|_{\mathcal{C}^{p,\beta}(\Gamma)}}_{\textit{truncation error}} + \underbrace{\widetilde{C}_{p,q,\beta} \left(\frac{h}{4r}\right)^{2q} \|\sigma\|_{\mathcal{C}^{2q,\beta}(\Gamma)}}_{\textit{quadrature error}}. \tag{2.7}$$

Theorem 2.1 shows there are mainly two sources of error for QBX: the truncation error from truncating (2.4) to (2.6), and the quadrature error when evaluating (2.5) using a quadrature rule. To control the quadrature error, we need to make sure the expansion center is adequately separated from the mesh. Specifically, Theorem 2.1 shows we need to satisfy $r > h/4$ to ensure that quadrature error decreases when increasing the quadrature order q. The quadrature order q and the QBX order p can be chosen based on the desired accuracy.

2.2 FMM ACCELERATION

For an expansion center associated with each target, calculating QBX local expansion coefficients in (2.5) requires $O(N)$ work, where N is the number of quadrature points. Therefore, for M target points, evaluating layer potential using QBX naively, as described in the last section, has complexity $O(MN)$. It has been shown [7] that QBX can be combined with Fast Multipole Method (FMM) by treating the QBX local expansion as a special kind of target. Different from the original FMM used for particle simulations [8] (called point FMM for the rest of this thesis), a local expansion is formed instead of evaluating its potential. This method lowers the overall cost to $O(M+N)$. Using similar ideas, recent work [9] develops a new algorithm called Geometric Global Accelerated QBX (GIGAQBX) by modifying

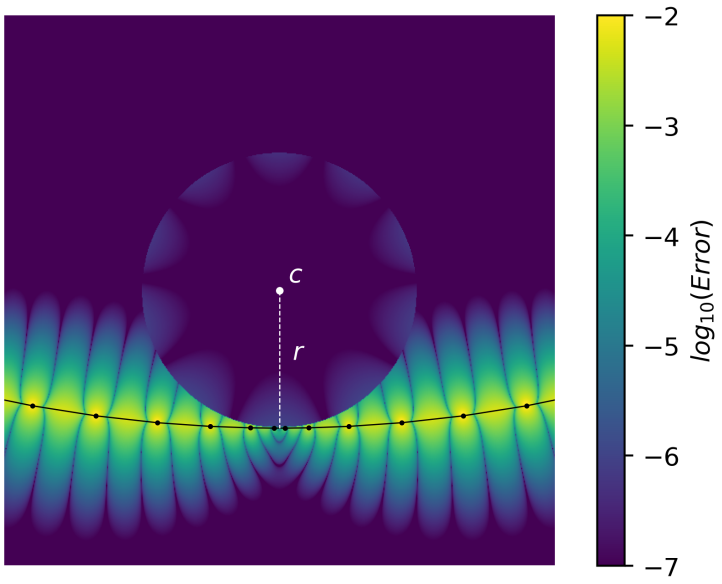


Figure 2.3: Error of evaluating the Single Layer Potential (2.1) using QBX for targets in a circle around the expansion center, overlaid on top of the error of using the Gaussian quadrature directly. The contour is the same circle as in Figure 2.1, but this figure only shows part of it. The QBX order is 6. The Gaussian quadrature used for calculating the QBX local expansion coefficients (2.5) and the target potentials outside the QBX expansion circle, has 20 panels and 10 quadrature points on each panel, which is the same as Figure 2.1b.

the FMM algorithm and provides rigorous error bound. This algorithm is summarized in the rest of this section.

Suppose we would like to compute the potentials of target points x_i due to source points y_j in the form of

$$x_i = \sum_j f(x_i, y_j)\sigma(y_j). \tag{2.8}$$

One of the core idea of FMM is we can summarize the source contributions inside a circle by a **multipole expansion** [8], which is accurate for targets outside the expansion. Similarly, we can form a **local expansion** which summarizes source contributions far away, and is accurate for targets inside the expansion. Then, when sources and targets are separated from each other, we can form a multipole expansion around the sources, translate the multipole expansion to a local expansion close to the targets, and evaluate the potentials using the local expansion. For M sources and N targets, the complexity of forming the multipole expansion is $O(M)$, the complexity of evaluating target potentials using the local expansion is $O(N)$, and the complexity of the multipole-to-local translation is constant. Overall, this strategy improves the complexity from $O(MN)$ in the naive algorithm to $O(M + N)$.

In general, we cannot assume sources and targets are separated, so another core idea of FMM is to evaluate nearby sources directly, while evaluate far-away sources through multipole-to-local translations. First, all particles are enclosed in the root box. Then, whenever a box has more particles than a preset threshold, the box is bisected along each coordinate. The result is a quadtree (2D) or octree (3D), where each particle (source or target) belongs to a leaf box in the tree. For a box b, let $P(b)$ represent the parent box of b, and $|b|$ represent the distance between the center of b to its edge. We will use $\overline{B_p}(r, c)$ to represent the l^p ball of radius r centered at c. For example, $\overline{B_\infty}(r, c)$ is a square centered at c with side length $2r$. We will use C_b to denote the center of box b. We define **source box** as a box containing sources, **target box** as a box containing targets, and **target-ancestor box** as a target box or an ancestor of a target box. We use $A(b)$ to represent the ancestors of box b, and $D(b)$ to represent the descendants of box b. Additionally, when a function takes a single box as argument, it will also take set of boxes as argument. For example, suppose Φ is a set of boxes, then $A(\Phi)$ represents the ancestors of *some* boxes in Φ. For a target-ancestor box b, boxes within $\overline{B_\infty}(3|P(b)|, C_{P(b)})$ are classified into four **interaction lists** $U(b)$, $V(b)$, $W(b)$, $X(b)$ based on their geometric relationships with b. Potential contributions from boxes in different interaction lists are computed differently to be more efficient while remain accurate. After tree and interaction lists have been constructed, each source box forms a multipole expansion accurate outside the box by summarizing the sources within the box. For a non-leaf box, the multipole expansion can be formed by translating and adding the multipole expansions of its children. Therefore, multipole expansions of the leaf boxes are propagated upwards using a post-order tree traversal to form multipole expansions of all non-leaf boxes. Then for each target-ancestor box b, a local expansion accurate within the box is formed by translating from the multipole expansions of boxes in $V(b)$ and source points of boxes in $X(b)$. These local expansions are then propagated downwards so that the local expansion of each box picks up potential contributions from source boxes outside $\overline{B_\infty}(3|P(b)|, C_{P(b)})$. Finally, the potential of each target point in box b is computed by adding the potentials calculated from source points in $U(b)$, from the multipole expansions of $W(b)$, and from the downward-propagated local expansion.

There are three main changes of GIGAQBX compared to traditional point FMM.

- **Target points may reside in non-leaf boxes**. Since QBX expansion centers are considered a special kind of target in GIGAQBX, targets may have non-zero size or "extent" associated with them. To ensure accuracy, GIGAQBX requires that a QBX center may reside in a box only when the center and its associated extent can fit inside this box's **target confinement region (TCR)**. The TCR of a box is a

region surrounding the box and enlarged by a pre-chosen constant factor, called **target confinement factor**, denoted by t_f. To be precise, TCR of a box b centered at c is defined as $\overline{B_\infty}(|b|(1+t_f), c)$ in 2 dimensions and $\overline{B_2}(\sqrt{3}|b|(1+t_f), c)$ in 3 dimensions [10]. If a QBX center cannot reside in a box because of this restriction, it must reside in an ancestor of this box that it fits.

- **The interaction lists of a target box contain boxes from a wider region.** In traditional point FMM, when evaluating the potentials in a target box b, contributions from source boxes outside $\overline{B_\infty}(3|P(b)|, C_{P(b)})$ are not included in the interaction list. Instead, these contributions are counted using downward propagation, i.e. translating the local expansion of b's parent to the local expansion of b. GIGAQBX adopts the same idea of downward propagation as the point FMM, but to ensure accuracy, the interaction lists contain boxes within $\overline{B_\infty}(5|P(b)|, C_{P(b)})$.

- W **list and** X **list are reclassified into "close" list and "far" list.** Because of the presence of the TCR, some source boxes classified as W and X are too close to use expansion acceleration. To ensure accuracy, each of the list W and X are further reclassified into two lists: the "close" list which needs to be evaluated directly, and the "far" list which uses expansion acceleration.

2.2.1 Tree Construction

When building the FMM tree, GIGAQBX considers quadrature nodes as sources, QBX expansion centers and target points evaluated without QBX as targets. Target points evaluated using QBX are omitted at the moment because their potentials are evaluated using the local expansion of their associated QBX centers. At the beginning of the tree construction, all particles reside within the root box. Whenever a box has more particles than a preset threshold, the box is bisected for each coordinate into 4 smaller boxes (in 2D) or 9 smaller boxes (in 3D). Note that since a QBX center has "extent", it might not be able to fit inside the child box's TCR, and therefore has to be kept in the parent box. The centers unable to fit inside child boxes do not count towards the threshold.

2.2.2 Interaction Lists

For a target-ancestor box b, GIGAQBX classifies boxes within $\overline{B_\infty}(5|b|, C_b)$ into different categories based on their geometry relationships with respect to b. When counting the potential contributions, different interaction lists represent different rules, based on accuracy

and efficiency. However, the interaction list definitions in GIGAQBX are different from the point FMM. We define the **neighborhood** of a box b as $\overline{B_\infty}(5|b|, C_b)$. A box is said to be **well-separated** from b if the box is outside the neighborhood of b. We define the **colleagues** of a box b as a set of boxes at the same level and within the neighborhood of b. The colleagues of b are denoted by $T(b)$. We use the same definition of **adequate separation relation** as [10], which is included here for completeness.

Definition 2.1. *We define a relation $\prec$ over the set of boxes and target confinement regions within the tree, with $a \prec b$ to be understood as 'a is adequately separated from b, relative to the size of a'.*

We write $a \prec TCR(b)$ for boxes a and b if the l^∞ (in 2D)/l^2 (in 3D) distance from the center of a to the boundary of $TCR(b)$ is at least $3|a|$.

We write $TCR(a) \prec b$ for boxes a and b if the l^∞ distance from the center of a to the boundary of b is at least $3|a|(1 + t_f)$.

We write $a \not\prec b$ to denote the negation of $a \prec b$.

[9] gives a precise definition for each interaction list in GIGAQBX. These definitions are included here for completeness. We refer readers to [9] for motivations on these interaction lists.

Definition 2.2. **U(b)**: *For a target box b, $U(b)$ consists of all leaf boxes in $D(b) \cup \{b\}$ and the set of boxes adjacent to b.*

Potential contributions from sources in $U(b)$ are computed directly without any expansion acceleration.

Definition 2.3. **V(b)**: *For a target-ancestor box b, $V(b)$ consists of the children of $T(P(b))$ that are well-separated from b.*

Note that boxes in $V(b)$ have the same size and level as b. Potential contributions from $V(b)$ are counted by translating the multipole expansion of each box in $V(b)$ to the local expansion of b.

Definition 2.4. **W(b)**: *For a target box b, a box $d \in D(T(b))$ is in $W(b)$ if d is not adjacent to b, and for all $w \in A(d) \cap D(T(b))$, w is adjacent to b.*

$W(b)$ contains boxes whose potential contributions cannot be evaluated accurately using multipole-to-local translation because b is too large. Instead, how the contributions from a box $d \in W(b)$ are counted is based on the adequate separation relation between d and TCR(b). If $d \prec TCR(b)$, the potential contribution from d can be counted by evaluating the

mutipole expansion of d at each target in b. The box d in this case is said to be in $W^{\text{far}}(b)$. If $d \not\prec \text{TCR}(b)$, d is too close to use any expansion acceleration, and the sources in d are counted directly, just like $U(b)$. In this case, d is said to be in $W^{\text{close}}(b)$. Note that for a box $p \in W^{\text{far}}(b)$, the multipole expansion of p contains the potential contributions from p's descendants, but for a box $q \in W^{\text{close}}(b)$, the direct evaluation does not count for sources in q's descendants. To pick up their contributions, for each box w of q's children, w needs to be added to either $W^{\text{close}}(b)$ or $W^{\text{far}}(b)$, depending on the adequate separation relation between w and $\text{TCR}(b)$. The precise definitions of $W^{\text{close}}(b)$ and $W^{\text{far}}(b)$ are as follows:

Definition 2.5. $\mathbf{W}^{close}(\mathbf{b})$*: For a target box b, a leaf box d is in $W^{close}(b)$ if $d \in D(W(b)) \cup W(b)$ such that $d \not\prec TCR(b)$.*

Definition 2.6. $\mathbf{W}^{far}(\mathbf{b})$*: For a target box b, a box d is in $W^{far}(b)$ if $d \in D(W(b)) \cup W(b)$ such that $d \prec TCR(b)$ and, for all $w \in A(d) \cap (D(W(b)) \cup W(b))$, $w \not\prec TCR(b)$.*

Definition 2.7. $\mathbf{X}(\mathbf{b})$*: For a target-ancestor box b, a leaf box d is in $X(b)$ if d is a colleague of b and d is not adjacent to b. Additionally, a leaf box d is in $X(b)$ if d is a colleague of some ancestor of b and d is adjacent to $P(b)$ but not b itself.*

When counting the potential contribution from a box $d \in X(b)$, multipole-to-local translation is not accurate because d is too large. Instead, how the potential can be counted depends on whether $\text{TCR}(b)$ and d are adequately separated. If $\text{TCR}(b) \prec d$, the potential contribution from d can be counted by adding each source in d to the local expansion of b. In this case, d is said to be in $X^{\text{far}}(b)$. If $\text{TCR}(b) \not\prec d$, d is too close to b and the potential contribution of d needs to be evaluated directly without expansion acceleration, just like $U(b)$. In this case, d is said to be in $X^{\text{close}}(b)$. Note that for a box $p \in X^{\text{far}}(b)$, the potential contribution of p is added to the local expansion of b, and can therefore propagate downwards to b's descendant. However, for a box $q \in X^{\text{close}}(b)$, the source contribution is counted directly and will not propagate to b's descendant, so each box w of b's children, q needs to be added to either $X^{\text{close}}(w)$ or $X^{\text{far}}(w)$, depends on the adequate separation relationship between $\text{TCR}(w)$ and q. To sum up, the precise definitions of $X^{\text{close}}(b)$ and $X^{\text{far}}(b)$ are as follows

Definition 2.8. $\mathbf{X}^{close}(\mathbf{b})$*: A box d is in $X^{close}(b)$ if $d \in X(w)$ for some $w \in A(b) \cup \{b\}$ and $TCR(b) \not\prec d$.*

Definition 2.9. $\mathbf{X}^{far}(\mathbf{b})$*: A box $d \in X(b)$ is in $X^{far}(b)$ if $TCR(b) \prec d$. Furthermore, if b has a parent, a box $d \in X^{close}(P(b))$ is in $X^{far}(b)$ if $TCR(b) \prec d$.*

Figure 2.4 shows these interaction lists in action for a target box in 2D.

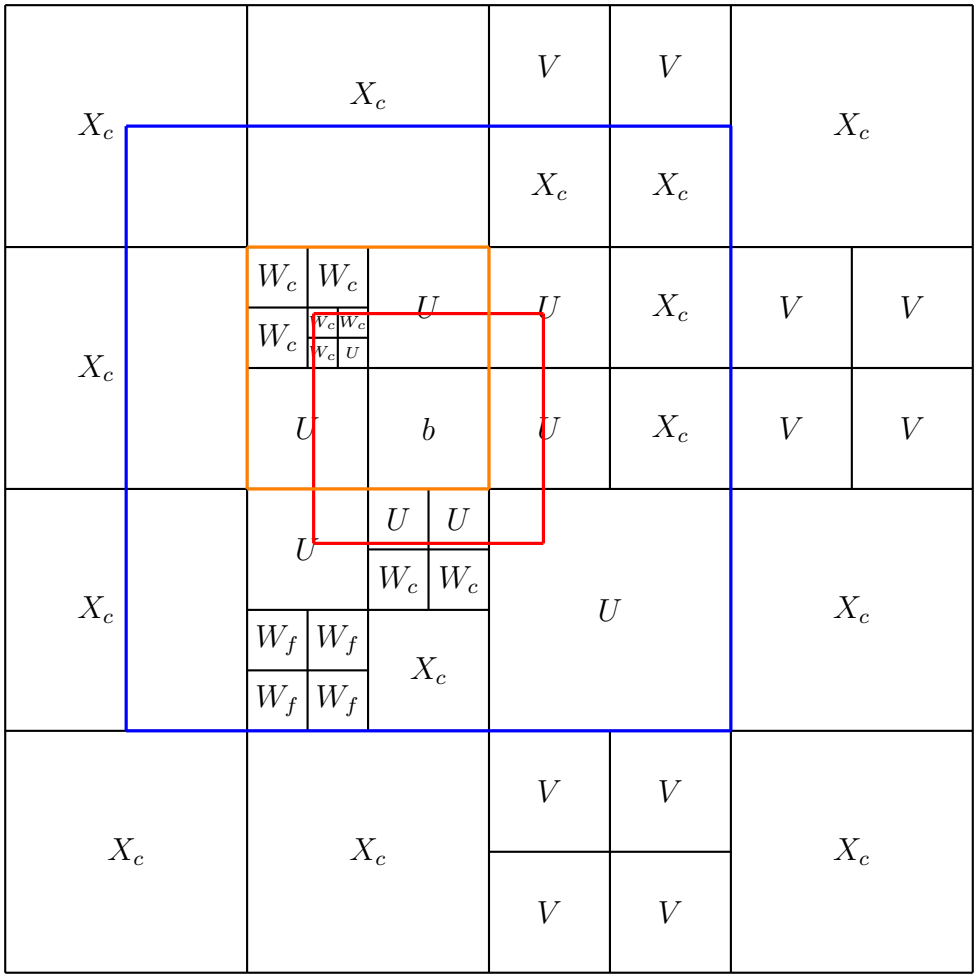


Figure 2.4: Interaction lists for a target box b.

This figure shows the interaction lists in 2D of a target box b. In this example, we choose $t_f = 0.9$, and the red box is the TCR of b. The blue box represents the neighborhood of b, which means every boxes inside the blue box with the same size of b are colleagues of b. The orange box is the parent of b, and the whole region shown in this figure is within the neighborhood of $P(b)$. The letters inside the boxes indicate the interaction lists. U means the box is in $U(b)$. V means the box is in $V(b)$. W_c means the box is in $W^{\text{close}}(b)$. W_f means the box is in $W^{\text{far}}(b)$. X_c means the box is in $X^{\text{close}}(b)$. X_f means the box is in $X^{\text{far}}(b)$.

2.3 COMPLETE GIGAQBX ALGORITHM

To facilitate the discussion of GIGAQBX algorithm, we use $\mathbb{U}$ to denote the set of all boxes, P_x to denote potential at target point x, M_b to denote the multipole expansion of box b, L_b to denote the local expansion of box b, and Q_c to denote the QBX local expansion

at an expansion center c.

Input: source mesh Γ, desired accuracy ϵ, source density μ, target points

Output: the approximated potential using QBX with FMM acceleration at each target point

Preliminary stage:

Choose the maximum number of particles per box $n_{\max}$ and the target confinement factor t_f.

Choose QBX order, FMM order and oversampled quadrature node count based on ϵ.

Discretize and refine the input mesh Γ according to [10], to obtain quadrature nodes of stage-2 discretization and on-surface target points.

Upsample the source density μ to the quadrature nodes, and assign each QBX target to an expansion center [10].

Initialize P_x to 0 for each target point x. Initialize M_b and L_b to 0 for each box b. Initialize Q_c to 0 for each QBX local expansion centered at c.

Stage 1: Build tree and interaction lists

Create a spatial tree from sources, targets and QBX expansion centers based on $n_{\max}$ and t_f, as described in Section 2.2.1.

Create interaction lists for all target-ancestor boxes, as described in Section 2.2.2.

Stage 2: Form multipoles

for each box $b \in \mathbb{U}$ **do**

Form a multipole expansion M_b at C_b from sources in b.

end for

Stage 3: Propagate multipole expansions upward

for each box $b \in \mathbb{U}$ with postorder traversal **do**

Translate M_b from C_b to $C_{P(b)}$, and add the translation result to $M_{P(b)}$.

end for

Stage 4: Evaluate direct interactions

for each box $b \in \mathbb{U}$ **do**

For each point target x owned by box b, compute potentials due to source points in $U(b)$, $W^{\text{close}}(b)$ and $X^{\text{close}}(b)$ directly and add the result to P_x.

end for

for each box $b \in \mathbb{U}$ **do**

For each QBX expansion center c owned by box b, add the contributions from source points in $U(b)$, $W^{\text{close}}(b)$ and $X^{\text{close}}(b)$ to Q_c.

end for

Stage 5: Translate multipole to local expansions

for each box $b \in \mathbb{U}$ **do**
  for each box $d \in V(b)$ **do**
    Translate multipole expansion M_d to local expansion of box b, and add the result to L_b.
  end for
end for

Stage 6: Evaluate multipole at targets

for each box $b \in \mathbb{U}$ **do**
  for each box $d \in W^{\text{far}}(b)$ **do**
    For each point target x owned by box b, evaluate the potential from the multipole expansion M_d, and add the result to P_x.
  end for
end for
for each box $b \in \mathbb{U}$ **do**
  for each box $d \in W^{\text{far}}(b)$ **do**
    For each QBX expansion center c owned by box b, translate the multipole expansion M_d to the QBX local expansion at c, and add the result to Q_c.
  end for
end for

Stage 7: Form locals from source points

for each box $b \in \mathbb{U}$ **do**
  For each source point y owned by boxes in $X^{\text{far}}(b)$, expand y to the local expansion of b and add the result to L_b.
end for

Stage 8: Propagate local expansions downward

for each box $b \in \mathbb{U}$ with preorder traversal **do**
  for each box d of b's children **do**
    Translate L_b from C_b to C_d, and add the translation result to L_d.
  end for
end for

Stage 9: Evaluate local expansions

for each box $b \in \mathbb{U}$ **do**
  For each point target x owned by b, evaluate the potential of x from the local expansion L_b, and add the potential result to P_x.

end for

for each box $b \in \mathbb{U}$ **do**

For each QBX expansion center c owned by b, translate the local expansion L_b to QBX local expansion at c, and add the result to Q_c.

end for

Stage 10: Evaluate QBX targets

for each QBX local expansion center c **do**

For each QBX target x associated with c, evaluate the potential of x using QBX local expansion Q_c, and store the result in P_x.

end for

CHAPTER 3: DISTRIBUTED ALGORITHM

This section will discuss the extension of GIGAQBX to distributed-memory systems. Our distributed algorithm performs the same computation as single-node GIGAQBX, albeit the order of the computation might be different. Therefore, the result of the distributed algorithm should be the same as the single-node one, within the tolerance of roundoff error. At the beginning of the algorithm, we assume the source geometry, source density and target points are available on the root rank. Then, the workload is distributed among worker ranks. Work partition is important in our algorithm because it influences the communication volume and the load balancing. Suppose there are n ranks in total, the root rank constructs a set partition $\{R_1, R_2, R_3, ..., R_n\}$ of all boxes, where $R_i \subseteq \mathbb{U}$, $\cup_{i=1}^{n} R_i = \mathbb{U}$, and $R_i \cap R_j = \emptyset$ for $i \neq j$. Each rank i is responsible for forming the multipole expansion of each box in R_i, computing the target potentials for each non-QBX target in boxes of R_i, forming the QBX local expansion of each QBX center in R_i, and computing the potential of each QBX target associated with a center in R_i. The work partition will be discussed in more detail in Section 3.1. After distributing the complete tree structure and subset of particles from the root rank to each worker rank according to the work partition, rank i can then form the multipole expansion of each box in R_i and propagate upwards. Note that an internal FMM tree node in general can have descendants from multiple ranks, so the propagated multipoles are only partial. Communication is needed for forming the complete multipole expansions and distributing the multipole expansions to ranks which use them for forming local expansions and computing potentials. The multipole communication will be covered in more detail in Section 3.2. Once the multipole communication is completed, each rank i has all dependencies for calculating the potentials in R_i and can follow stage 4–10 discussed in Section 2.3, in a manner identical to the single-node algorithm. Finally, all worker ranks send the computed potentials to the root rank. The complete distributed algorithm is given in Section 3.3.

3.1 WORK PARTITION

When designing a work partition scheme, we consider two criteria: communication volume and load balancing.

For distributed point FMM algorithms on N ranks, [3] uses a space-filling curve to sort boxes in Morton order, and cuts the one-dimension list into N pieces. The benefit of using the space-filling curve is that it keeps each partition largely spatially grouped, and therefore

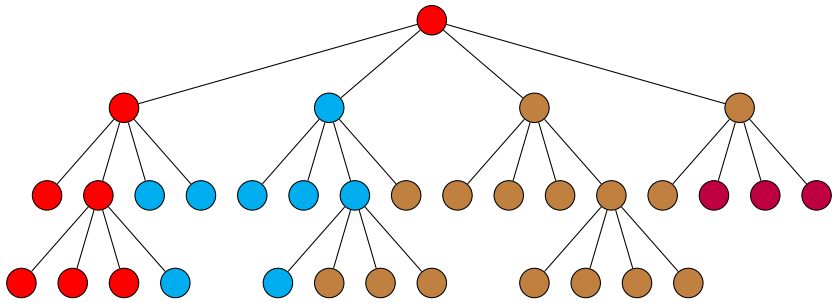


Figure 3.1: Partition of the tree along pre-order depth-first traversal. Each circle in this figure represents an internal or leaf box. The circle at the top represents the root box, while the circles at the bottom represent the leaf boxes. The edge between two circles represents the parent-child relation between the two boxes. In this figure, there are four ranks in total, where circles with different colors represent boxes owned by different partitions.

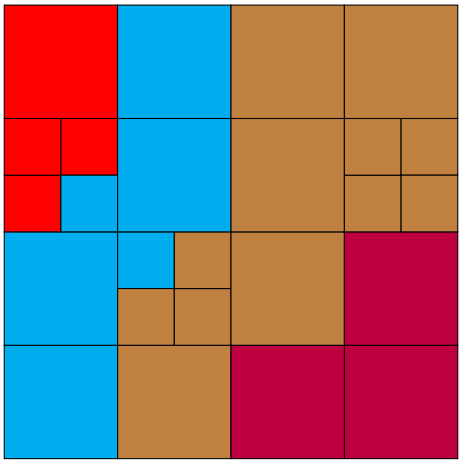


Figure 3.2: The same partition as Figure 3.1 in spatial view among leaf boxes. This figure shows each partition is spatially grouped when partitioning along depth-first traversal.

reduces communication since the communication across ranks happens at the boundary of domain. Our distributed GIGAQBX algorithm adopts a similar approach. One difference between GIGAQBX compared to traditional point-FMM algorithms is that targets can reside in the internal nodes in addition to the leaf nodes. To accommodate this difference but also maintain spatial locality of the groups, our algorithm partitions the nodes along the pre-order depth-first traversal of the tree. Figure 3.1 shows one example of such a partitioning. Figure 3.2 shows the same partitioning in the spatial view among leaf boxes, and demonstrates each partition is spatially grouped. Note that Figure 3.2, unlike figure 3.1, does not show partition membership of parent boxes.

In order to improve load balancing, the cut-off points along the depth-first order are chosen by partitioning the boxes evenly, weighed by the cost of each box. The running time

of a box in a particular stage is estimated by a cost model [11]. Our implementation uses the leading-order cost, calibrated through a sample of small test cases. For example, the leading order cost of a target box b due to sources in a box $u \in U(b)$ is proportion to the number of sources in u times the number of targets in b. Therefore, the calibration parameter β of U list can be estimated as

$$\beta = \frac{\text{total running time of } U \text{ lists in the test run}}{\sum_{b\in\mathbb{U}} \sum_{u\in U(b)} (\text{number of targets in } b) \cdot (\text{number of sources in } u)}. \tag{3.1}$$

The cost of a new box b' during a distributed run due to list U can then be estimated as

$$\text{cost}(b') \approx \beta \cdot (\text{number of targets in } b') \sum_{u\in U(b')} (\text{number of sources in } u). \tag{3.2}$$

The cost of other stages can be estimated in a similar way. The total cost of each box is then estimated by adding the estimated costs of all stages. The accuracy of the cost model is assessed in Section 4.4.

3.2 MULTIPOLE COMMUNICATION

When rank i propagates the multipole expansion upwards, it is possible that a box $b \in A(R_i)$ has descendants that are owned by a different rank. Therefore, in general the propagated multipole expansion of b at rank i is only partial, and communication is needed between rank i and j to compute the complete multipole expansion of a box b. In this case, we call both rank i and rank j contributors of the box b. Furthermore, a box $d \in R_k$ may depends on the multiple expansion of b. For example, b might be in $V(d)$ and rank k will perform the multipole-to-local translation from b to d. In this case, communication is also required to send the complete multipole expansion of b to rank k, and we call rank k a user of the box b. These two cases for multipole communication are illustrated in Figure 3.3. The precise definitions of contributors and users are adapted from [4], and given as follows:

Definition 3.1. *Rank i is a **contributor** of box b if $b \in R_i \cup A(R_i)$.*

Definition 3.2. *Rank i is a **user** of box b if there is a box $p \in R_i$ such that $b \in W^{far}(p)$, or there is a box $q \in R_i \cup A(R_i)$ such that $b \in V(q)$.*

The easiest way to implement the multipole communication is through the MPI-specified all-reduce algorithm, which adds values from each rank, and distributes the result to all ranks. However, for a particular box, few ranks are users and few ranks contribute. This means

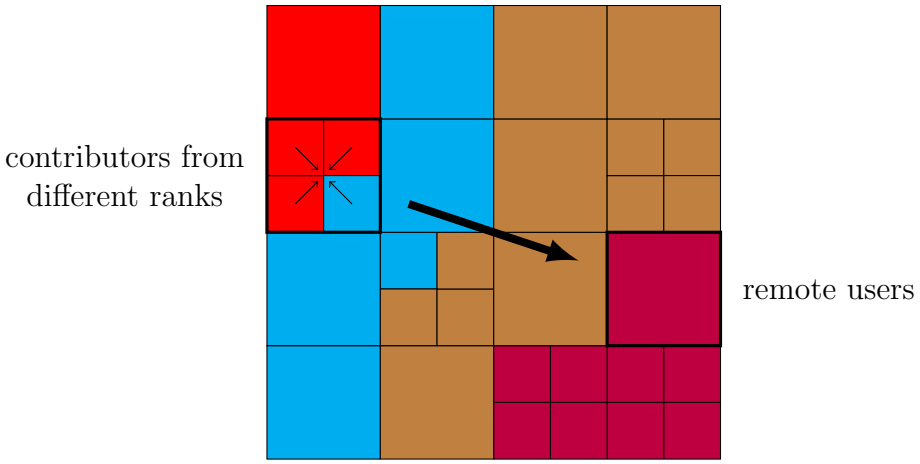


Figure 3.3: Multipole expansion communication. This figure demonstrates the two reasons why communications are needed for multipole expansions. First, the box on the left is both an ancestor of boxes owned by the red rank, and an ancestor of boxes owned by the blue rank. After the upward propagation, both the red rank and the blue rank have only the partial multipole expansions of the box. Therefore, communication is needed to add the partial expansions together to form the complete multipole expansion. Second, suppose the box on the left is in list V of the box on the right, and the box on the right is owned by the purple rank. Then, communication is needed to send the complete multipoe expansion of the box on the left to the purple rank for the multipole-to-local translation.

the input multipole vector on each rank is sparse, and the results are distributed to ranks which do not need them, so using all-reduce will be wasteful in terms of communication volume. Another way to implement multipole communication is to delegate to the owning rank, i.e. for an internal box b with contributors from multiple ranks, rank r with $b \in R_r$ communicates with all b's contributors, sums the partial multipole expansions up, and distributes the complete multipole expression to all b's users. Although this approach reduces the communication volume compared to the all-reduce approach, it could lead to unfavorable communication pattern. Consider a box $b \in R_r$ close to the root of the tree, b could have many contributors and users. This means r needs to communicate with most of the ranks.

To implement an efficient scheme for multipole communication, we adopt and generalize the tree-based strategy proposed in [4]. This algorithm is summarized in Algorithm 3.1. In the algorithm, we use $T(k)$ to represent the set of boxes which rank k is a user of, i.e. $T(k) = W^{\text{far}}(R_k) \cup V(R_k \cup A(R_k))$. In Algorithm 3.1, rank r maintains the set S which represents the set of boxes whose multipole expansions are contributed or received by rank r. At each stage i, rank r chooses a partner p, and sends multipole expansions of boxes of which at least one rank in $[p_s, p_e]$ is a user. Once the current stage is finished, rank r does not need to communicate with $[p_s, p_e]$ again. Figure 3.4 shows an example of this communication

pattern for 8 ranks.

Algorithm 3.1 Tree-based Multipole Communication [4]

Assume the current rank is r, and the total number of ranks is 2^d
Input: partial multipole expansions from sources in R_r
Output: complete multipole expansions of all boxes of which rank r is a user
$S \leftarrow R_r \cup A(R_r)$
for $i \leftarrow d-1$ to 0 **do**
  $p \leftarrow r$ XOR 2^i
  $p_s \leftarrow p$ AND $(2^d - 2^i)$
  $p_e \leftarrow p$ OR $(2^i - 1)$
  send multipoles of $S \cap \left(\cup_{k=p_s}^{p_e} T(k)\right)$ to p
  receive multipoles from p and reduce
  append received boxes to S
end for

3.3 COMPLETE DISTRIBUTED ALGORITHM

Input: source geometry Γ, source density μ and target points on the root rank

Output: computed potentials of all target points on the root rank

Assume: t_f, QBX order, FMM order and quadrature order have been chosen. The cost model has been calibrated according to Section 3.1.

Step 1: Construct tree on the root rank

The root rank

- discretizes and refines the source geometry Γ, builds the complete FMM tree and interaction lists, and upsamples the source density μ to quadrature nodes, in the same way as the single-node algorithm described in the preliminary stage of Section 2.3
- merges interaction lists W^{close} and X^{close} into U
- estimates the cost of each box and computes the box partitions $\{R_1, R_2, ..., R_n\}$, as described in Section 3.1

Step 2: Distribute the tree and particles

The root rank

- distributes the complete tree structure without particles to all ranks
- sends non-QBX targets in R_i, expansion centers in R_i, and QBX targets associated with expansion centers in R_i to rank i
- sends sources in $U(R_i)$ and $X^{\text{far}}(R_i)$ to rank i

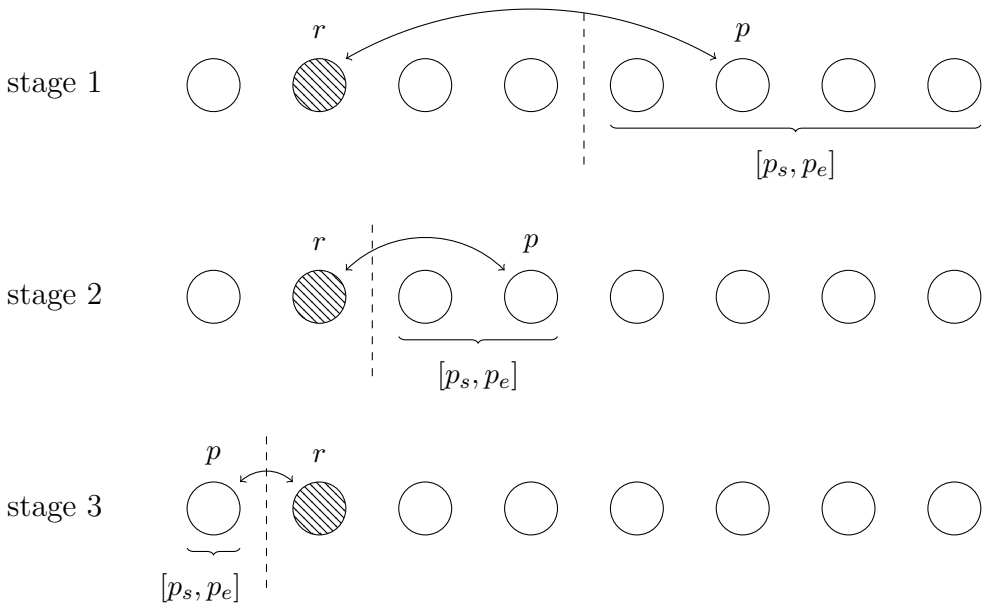


Figure 3.4: Tree-based multipole communication pattern for 8 ranks. Each circle in this figure represents one rank. This figure demonstrates Algorithm 3.1 from the point of view of rank r, which is represented by the shaded circle in the Figure. In this example, the communication is divided into three stages. At the first stage, all ranks are remaining for communication from rank r. During the first stage, rank r sends multipoles needed by $[p_s, p_e]$ to rank p, so at the second stage, only the first four ranks are remaining for communication from r. Similarly, at the last stage, only the first two ranks are remaining. For each stage, the dashed line shows the division of the remaining ranks into two halves. $[p_s, p_e]$ represents the opposite half from rank r.

Each rank i independently

- receives the complete tree structure and the subset of particles from the root (for the rest of the algorithm, we will use *local tree* to refer to this tree with the subset of particles on each rank)
- constructs the interaction lists of the local tree

Step 3: Distribute source densities

Distribute densities associated with sources in $U(R_i)$ and $X^{\text{far}}(R_i)$ from the root rank to rank i.

Step 4: Compute multipole expansions

Each rank i independently

- forms multipole expansions of boxes in R_i from sources, like stage 2 in Section 2.3
- propagates the multipole expansions upwards to form the partial multipole expansions of boxes in $R_i \cup A(R_i)$, like stage 3 in Section 2.3

Step 5: Multipole communication
All ranks collectively communicate the partial multipole expansions to form the complete multipole expansions, and distribute the complete multipole expansions to all users, using the tree-based approach discussed in Section 3.2.

Note that after this step, each worker rank i has all dependencies needed to compute target potentials and form QBX local expansions in R_i.

Step 6: Compute local expansions
Each rank i independently

- forms the local expansions of $R_i \cup A(R_i)$ from V and X^{far} lists, like stage 5 and 7 in Section 2.3
- propagates the local expansions of $R_i \cup A(R_i)$ downwards, like stage 8 in Section 2.3

Note that after this step, each worker rank i contains the complete local expansion for boxes in R_i.

Step 7: Evaluate target potentials
Each rank i independently

- evaluates potentials of non-QBX targets in R_i from list U (stage 4 in Section 2.3), list W^{far} (stage 6 in Section 2.3), and the owning box's local expansion (stage 9 in Section 2.3)
- forms QBX local expansions for centers in R_i from list U (stage 4 in Section 2.3), list W^{far} (stage 6 in Section 2.3), and the owning box's local expansion (stage 9 in Section 2.3)
- evaluates potentials of QBX targets from their associated QBX local expansions, like stage 10 in Section 2.3

Step 8: Gather computed potentials
Each rank i sends the computed potentials to the root rank. The root rank then merges the received potentials into the output buffer.

CHAPTER 4: EVALUATION

In this chapter, we will evaluate various aspects of our distributed algorithm and its implementation. Section 4.4 assesses the accuracy of the cost model. Section 4.5 shows the scaling of a fixed problem with increasing number of compute nodes. Section 4.6 verifies that the implementation scales linearly with respect to the number of sources and targets.

4.1 PROBLEM SETUP

The tests throughout this chapter evaluate the single layer potential (1.5), with the three-dimensional Laplace kernel. In other word, the kernel function G in (1.5) is the solution to

$$\nabla^2 G = \delta, \tag{4.1}$$

whose closed-form is

$$G(x, y) = -\frac{1}{4\pi} \cdot \frac{1}{|x - y|}. \tag{4.2}$$

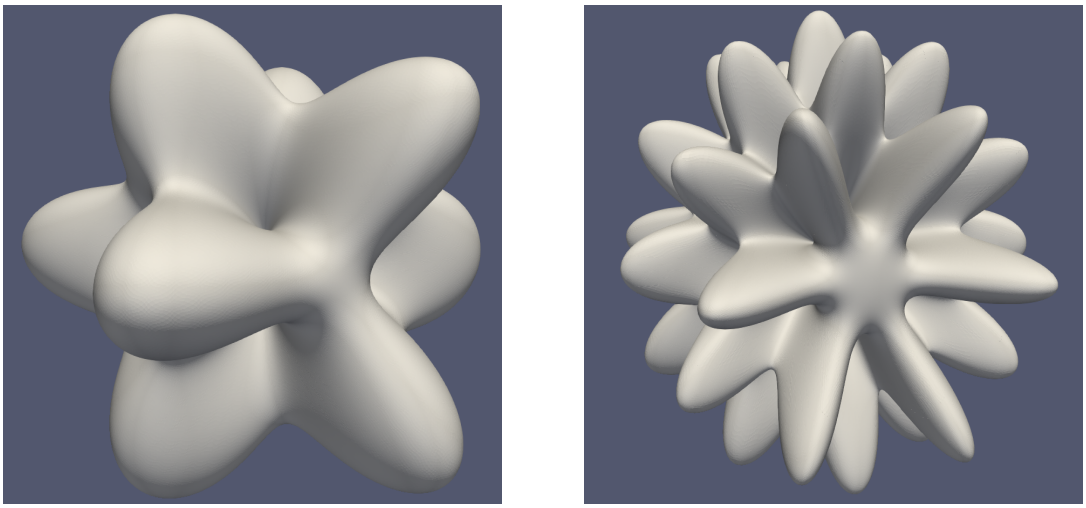
The geometry Γ is generated by warping a sphere made of triangles with radius in spherical coordinate system equal to the spherical harmonics,

$$Y_n^m(\theta, \phi) = \sqrt{\frac{2n+1}{4\pi} \frac{(n-m)!}{(n+m)!}} P_n^m(\cos(\theta)) e^{im\phi}, \tag{4.3}$$

where P_n^m are the associated Legendre polynomials. We refer to the geometry generated as "urchin". Plots of these "urchin" geometries with different m, n parameters are shown in Figure 4.1. All target points are located on the geometry surface.

4.2 IMPLEMENTATION DETAILS

We primarily implemented our algorithm in Python. Tree construction, interaction list construction and local tree filtering on the root rank are implemented in OpenCL, and incorporated via PyOpenCL [12]. Inter-node communication is handled by MPI, through mpi4py [13]. We wrap FMMLIB [14] in OpenMP for local translation operators.



(a) $m = 3, n = 5$ (b) $m = 6, n = 11$

Figure 4.1: "urchin" geometry used for evaluating the distributed algorithm

4.3 MACHINES

The tests in this chapter are performed on the Comet supercomputer in San Diego Supercomputer Center. Each compute node of Comet has 2 Intel Xeon E5-2680v3 CPUs on 2 sockets, and 128 GB of DDR4 DRAM. The compute nodes are connected with Infiniband FDR interconnects, with 56Gb/s bidirectional bandwidth on each node.

4.4 COST MODEL ACCURACY

To assign similar amount of computational work to each worker rank, our distributed algorithm uses a cost model to predict the cost of each box. Therefore, the accuracy of the cost model directly influences the load balancing, and hence the scalability. In this section, we evaluate the accuracy of the cost model. We first calibrate the cost model by evaluating the layer potential described in Section 4.1 on six 'urchin' geometries: two of them are generated using $m = 7, n = 13$, two of them are generated using $m = 8, n = 15$, and two of them are generated using $m = 9, n = 17$. We then evaluate the layer potential on two new 'urchin' test geometries. One of the new geometries is generated using $m = 6, n = 11$, smaller than the geometries used for calibration. The other new geometry is generated using $m = 10, n = 19$, larger than the geometries used for calibration. For each new test geometry, we report the predicted cost using the calibrated cost model, compared to the actual running time on Comet. The test result for the smaller test geometry is listed in Table 4.1, while the result for the larger test geometry is listed in Table 4.2.

stage	actual time	predicted time	error
form multipoles (stage 2)	17.09s	17.51s	2.5%
upward propagation (stage 3)	4.28s	4.88s	14.0%
point-to-QBX-local (stage 4)	494.36s	494.48s	0.0%
multipole-to-local (stage 5)	8.69s	8.73s	0.4%
multipole-to-QBX local (stage 6)	128.47s	122.02s	5.0%
point-to-local (stage 7)	18.66s	18.70s	0.2%
downward propagation (stage 8)	4.42s	4.72s	6.8%
local-to-QBX-local (stage 9)	2.66s	2.61s	2.1%
evaluate QBX targets (stage 10)	23.54s	28.10s	19.4%

Table 4.1: Cost model accuracy for 'urchin' geometry with $m = 6, n = 11$. The stage numbers listed are referencing Section 2.3.

stage	actual time	predicted time	error
form multipoles (stage 2)	76.72s	75.12s	2.1%
upward propagation (stage 3)	20.39s	20.54s	0.7%
point-to-QBX-local (stage 4)	2897.29s	2818.32s	2.7%
multipole-to-local (stage 5)	37.69s	37.76s	0.2%
multipole-to-QBX local (stage 6)	683.05s	688.56s	0.8%
point-to-local (stage 7)	81.13s	81.63s	0.6%
downward propagation (stage 8)	19.60s	19.81s	1.1%
local-to-QBX-local (stage 9)	12.22s	12.35s	1.0%
evaluate QBX targets (stage 10)	136.71s	133.20s	2.6%

Table 4.2: Cost model accuracy for 'urchin' geometry with $m = 10, n = 19$. The stage numbers listed are referencing Section 2.3.

For the larger test case, Table 4.2 suggests the cost model is very accurate, with the relative errors of all stages below 3%. However, Table 4.1 indicates the cost model is less accurate when predicting the cost on smaller geometry for some stages.

4.5 STRONG SCALING

We perform experiments with various number of nodes to evaluate the scalability of our distributed algorithm. The problem is to evaluate a single layer potential with the Laplace

# nodes	# cores	time	speedup	efficiency
1	24	3800.1s	1.00x	100.0%
2	48	1978.5s	1.92x	96.0%
4	96	982.5s	3.87x	96.7%
8	192	503.7s	7.54x	94.3%
16	384	254.5s	14.93x	93.3%
32	768	131.4s	28.92x	90.4%
64	1536	70.9s	53.60x	83.7%

Table 4.3: Scaling of the distributed algorithm with fixed problem size

kernel on an "urchin" geometry with $m = 10$ and $n = 19$, as discussed in Section 4.1. In total, there are $46,104,998$ sources and $16,275,600$ targets. Among these targets, $10,850,400$ are expansion centers, $5,425,200$ are QBX targets. Since all targets reside on the surface, there are no non-QBX targets. The time measured is for the evaluation steps only, i.e. Step 3–8 in Section 3.3. We consider the mesh refinement and tree build a setup time because solving an integral equation using GMRES performs mesh refinement and tree build only once, but needs to evaluate the layer potential in each iteration. Suppose $T(n)$ represents the time taken when running on n nodes. The speedup of n nodes is defined as $\frac{T(1)}{T(n)}$, and the efficiency is defined as $\frac{T(n)}{nT(1)}$. The scaling result is listed in Table 4.3.

To gain more insights into the performance, Figure 4.2 breaks down the total evaluation time into four major components.

- forming multiple expansions and upward propagation (Step 4 in Section 3.3)
- tree-based multipole expansion communication (Step 5 in Section 3.3)
- forming local expansions and downward propagation, forming QBX local expansions, and evaluating the potentials of QBX targets and non-QBX targets (Step 6–7 in Section 3.3)
- load imbalance, which is defined as the difference between the average time and the maximum time of Step 6–7 across all ranks

Among these major components, the most costly is the third component above, i.e. Step 6–7 in Section 3.3. The majority of the time in this component spends on forming QBX local expansions from list U. Notice that each QBX local expansion is used for evaluating only one QBX target. [11, Section 2.7] shows we could optimize this operation by using *target-specific QBX expansion* (TSQBX). For each source-target pair, TSQBX lowers the complexity from $O((p+1)^2)$ to $O(p+1)$.

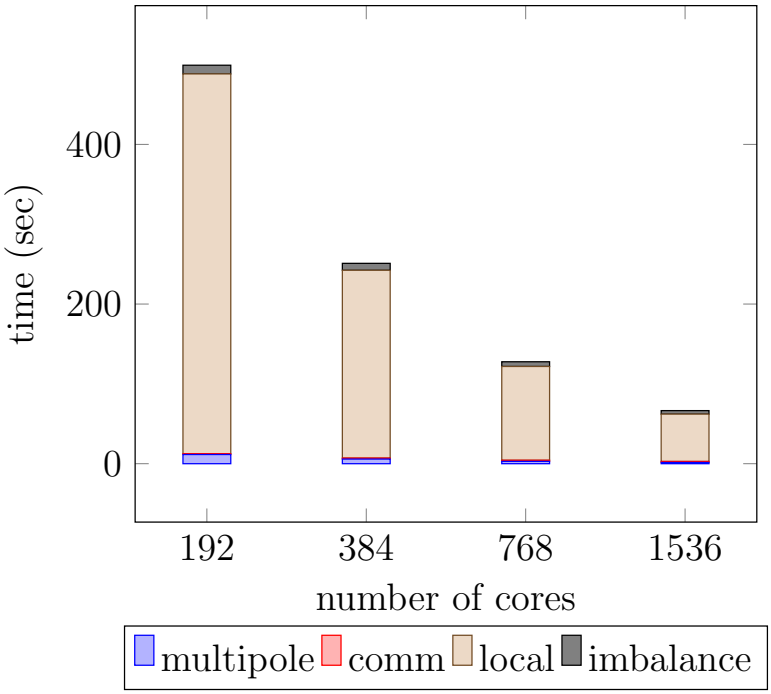


Figure 4.2: Time breakdown of each step for different machine sizes. In this figure, 'multipole' means the average time taken for forming multipole expansions from sources and propagating them upwards, which is Step 4 in Section 3.3. 'comm' means the time taken for the tree-based multipole communication, which is Step 5 in Section 3.3. 'local' means the average time taken for local operations like forming local expansions, forming QBX local expansions and evaluating target potentials, which is Step 6–7 in Section 3.3. 'imbalance' means the extra time from load imbalance, which is defined as the difference between the average time and the maximum time of Step 6–7 across all ranks.

Table 4.3 shows the distributed algorithm loses 16.3% of efficiency when the number of nodes reaches 64. There are two main reasons limiting scalability.

- Load imbalance is more pronounced as the number of nodes increases, as shown in Figure 4.3. One possible explanation for the load imbalance is, although the test geometry used in this section is the same as the larger geometry tested in section 4.4, for which the cost model has been shown to be accurate in Table 4.2, when the number of ranks increases, the local tree on each rank becomes smaller, and Table 4.1 indicates the cost model could be less accurate for smaller geometries.

- The costs of distributing source densities from the root rank to worker ranks (Step 3 in Section 3.3) and collecting the computed potentials from worker ranks to the root ranks (Step 8 in Section 3.3) remains the same as the number of nodes increases.

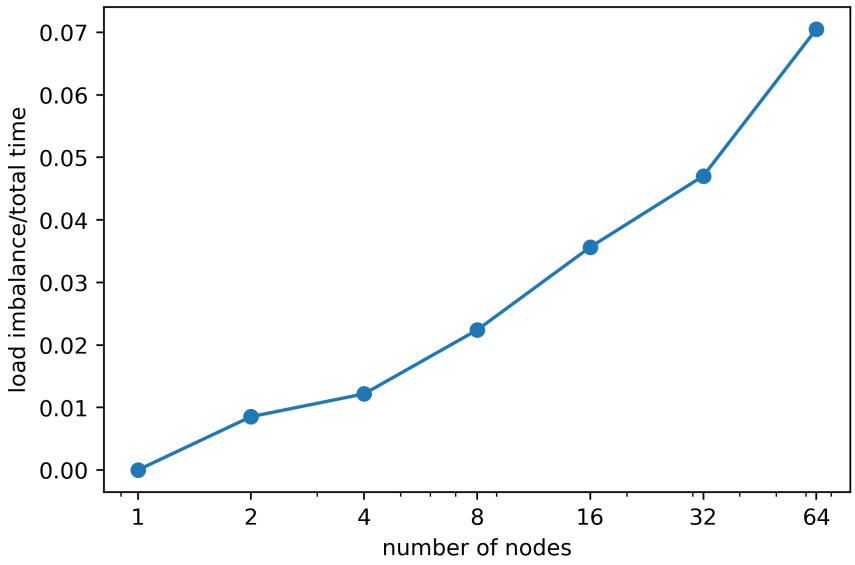


Figure 4.3: Semi-log plot of load imbalance when the number of nodes increases. The x-axis of this plot is the number of nodes in log scale. The y-axis is the proportion of the extra time due to load imbalance compared with the total local computation time (Step 6–7 in Section 3.3). The extra load imbalance time is defined as the difference between the average time and maximum time across all ranks.

m	n	number of sources	number of targets	time
4	7	3.98M	1.23M	231.05s
6	11	10.75M	3.43M	692.46s
8	15	24.20M	7.96M	1748.63s
10	19	46.10M	16.28M	3800.08s

Table 4.4: Scaling of GIGAQBX with various 'urchin' geometries

4.6 SCALING WITH PROBLEM SIZE

Section 2.2 claims QBX coupled with FMM has complexity $O(m+n)$, where m is the number of sources and n is the number of targets. This section verifies this claim when computing the on-surface layer potentials with 'urchin' geometries. We vary the problem size by changing the parameters m and n when generating the 'urchin' geometries, as discussed in Section 4.1. The evaluation is run on a single node. The result is listed in Table 4.4.

Assume the time taken obeys the linear model

$$\alpha \cdot (\text{number of sources}) + \beta \cdot (\text{number of targets}). \tag{4.4}$$

With the four cases listed in Table 4.4, we use a linear least square to obtain α and β. Figure

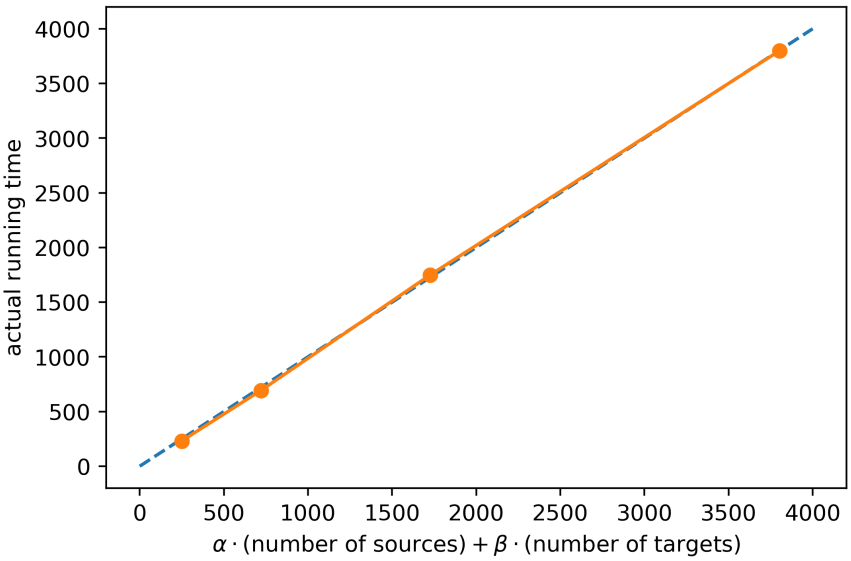


Figure 4.4: Time computed by the linear model using α and β compared to the actual running time. In this figure, the x-axis is the time computed using the linear model, and the y-axis is the actual running time. The orange line plots the modeled time and actual time for cases in Table 4.4. The blue dashed line is the identity function.

4.4 shows the linear model fits data in Table 4.4 well, suggesting the actual running time is linear in the number of sources and targets.

CHAPTER 5: DISCUSSIONS

5.1 CONCLUSIONS AND CONTRIBUTIONS

When solving boundary value problems of partial differential equations using integral equation methods, it is critical to evaluate layer potentials, usually with singular kernels, efficiently and accurately. QBX has been shown to be a viable way to compute layer potentials accuracy close to or on the boundary. When coupled with the GIGAQBX fast algorithm, we have verified QBX scales linearly with the problem size. In this thesis, we have shown an algorithm for extending QBX to distributed-memory systems. The algorithm constructs the complete tree on the root rank and then distributes it among worker ranks. Each rank then computes one partition of the total workload. In terms of accuracy, the distributed algorithm computes the same result as the single-node algorithm, barring rounding errors stemming from performing floating-point operations with different order. In terms to efficiency, this thesis has shown the distributed algorithm retains the efficient linear scaling with increasing problem size, and exhibits good scaling efficiency with increasing node count.

5.2 FUTURE IMPROVEMENTS

One major limitation of our current implementation is we refine the mesh and construct the complete tree on the root rank before distributing it. This strategy makes sense when the number of ranks is small, since usually the evaluation phase is more costly than the tree construction. Furthermore, when solving integral equations with GMRES, the algorithm performs mesh refinement and tree construction only once, while it performs the evaluation phase, the focus of this thesis, at each iteration. Nevertheless, when the number of nodes increases, the evaluation time decreases in proportion but the mesh refinement and tree build time would remain the same, which will eventually lead to unsatisfactory scalability. In the future, the input geometry of the algorithm should be distributed across ranks, where each rank contains only a subset of the geometry. The ranks should then refine the mesh in coordination, and only the local tree, described in Step 2 of Section 3.3, should be maintained on each rank. The evaluation phase can then follow the same algorithm as discussed in this thesis.

Figure 4.2 shows our current distributed algorithm is dominated by the time of local translation operators independently performed on each rank. Specifically, the cost of on-surface evaluation for Laplace kernel in three-dimension, as discussed throughout Chapter

4, is dominated by translating from each source in U list to the QBX local expansion. Section 4.5 discussed one way to optimize the local computation cost by using TSQBX to lower the complexity for each source-target pair. Using TSQBX only changes the handling of U list, W^{close} and X^{close} within stage 4 of Section 2.3, while the overall structure of the GIGAQBX algorithm remains the same. Therefore, our distributed algorithm only needs to make a similar change within Step 7 of Section 3.3 to leverage TSQBX. Another way to optimize the local computation is through GPU acceleration. Compared to CPUs, GPUs usually have higher arithmetic computation capacity through very high SIMD width. Local operators of GIGAQBX are computationally heavy and exhibit a high level of parallelism, making them ideal for running on GPUs. [4] applies GPU acceleration for a distributed point-fmm, and observes good speedup compared to CPU-only implementation.

As illustrated in Figure 4.3, load imbalance becomes more pronounced as the number of nodes increases. When the problem size is fixed and the number of nodes increases, the local tree on each node decreases in size. Table 4.1 suggests the cost model is less accurate when the geometry is smaller. This observation could explain the increasing load imbalance. To improve accuracy, we can use more complicated cost model. For example, instead of the leading order term only, the cost model could incorporate other significant terms, or even the constant term.

REFERENCES

[1] R. Kress, V. Maz'ya, and V. Kozlov, *Linear Integral Equations*. Springer, 1989.

[2] Y. Saad and M. H. Schultz, "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM Journal on scientific and statistical computing*, vol. 7, no. 3, pp. 856 – 869, 1986.

[3] M. S. Warren and J. K. Salmon, "A Parallel Hashed Oct-Tree N-body Algorithm," in *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, 1993.

[4] I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, and G. Biros, "A massively parallel adaptive fast-multipole method on heterogeneous architectures," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. IEEE, 2009.

[5] "*NIST Digital Library of Mathematical Functions*," http://dlmf.nist.gov/, Release 1.0.25 of 2019-12-15, f. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller, B. V. Saunders, H. S. Cohl, and M. A. McClain, eds. [Online]. Available: https://dlmf.nist.gov/10.23.E7

[6] A. Klöckner, A. Barnett, L. Greengard, and M. ONeil, "Quadrature by expansion: A new method for the evaluation of layer potentials," *Journal of Computational Physics*, vol. 252, pp. 332 – 349, 2013.

[7] M. Rachh, A. Klöckner, and M. O'Neil, "Fast algorithms for Quadrature by Expansion I: Globally valid expansions," *Journal of Computational Physics*, vol. 345, pp. 706 – 731, 2017.

[8] L. Greengard and V. Rokhlin, "A Fast Algorithm for Particle Simulations," *Journal of Computational Physics*, vol. 73, pp. 325–348, 1987.

[9] M. Wala and A. Klöckner, "A fast algorithm with error bounds for Quadrature by Expansion," *Journal of Computational Physics*, vol. 374, pp. 135–162, 2018.

[10] M. Wala and A. Klöckner, "A fast algorithm for Quadrature by Expansion in three dimensions," *Journal of Computational Physics*, vol. 388, pp. 655–689, 2019.

[11] M. Wala and A. Klöckner, "Optimization of fast algorithms for global Quadrature by Expansion using target-specific expansions," *Journal of Computational Physics*, vol. 403, 2020.

[12] "PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation," *Parallel Computing*, vol. 38, no. 3, pp. 157 – 174, 2012.

[13] L. Dalcín, R. Paz, and M. Storti, "MPI for Python," *Journal of Parallel and Distributed Computing*, vol. 65, no. 9, pp. 1108 – 1115, 2005.

[14] Z. Gimbutas and L. Greengard, "Computational Software: Simple FMM Libraries for Electrostatics, Slow Viscous Flow, and Frequency-Domain Wave Propagation," *Communications in Computational Physics*, vol. 18, no. 2, pp. 516 – 528, 2015.